STANDARD MUMPS
POCKET GUIDE

E2)

$E("ABC

E2,E3)

$E(VAR,3

STANDARD MUMPS POCKET GUIDE

INTRODUCTION

This booklet provides a concise summary of Standard MUMPS, and includes explanatory text and programming examples. This guide is therefore intended to provide an introduction to MUMPS programming for people not previously aware of the scope of MUMPS. Furthermore it provides a useful summary of Standard MUMPS for people currently programming in one of the several nonstandard MUMPS dialects, thereby indicating which features of their current dialect they should avoid in order to facilitate the later translation of their applications into Standard MUMPS.

This document summarizes all the commands, functions, operators and other features of Standard MUMPS and is believed to conform to Standard MUMPS as defined by the MUMPS Development Committee and the National Bureau of Standards. However, the reader is strongly cautioned against using this Pocket Guide as a substitute for the official MDC/ANSI description of Standard MUMPS. For details of the Standard, consult: O'Neill, J. T., Editor, MUMPS Language Standard, ANSI X11.1-1977, published by the MUMPS Development Committee, 1977; or Conway, M. E., MUMPS Programmers' Reference Manual, published by the MUMPS Development Committee, 1976. For instruction in advanced programming techniques, the following is recommended: Beckley, R. E., and Bridger, D. A., Advanced Mumps Techniques, published by MUG, 1977.

To obtain copies or price lists of the MUG and MDC documents or to obtain additional copies of this Guide, contact Mr. Richard Zapolin, MUMPS User's Group, MITRE Corporation, P. O. Box 208, Bedford, Massachusetts 01730.

Throughout this handbook, certain portions of the text are enclosed in brackets. This convention is used to denote certain limitations which are imposed not by the definition of Standard MUMPS, but by a concern for the portability of MUMPS programs from one implementation to another. These Portability Requirements should be met in order to facilitate this interchange.

ACKNOWLEDGMENTS

This second edition, compiled by Joel Achtenberg, constitutes an expansion and revision of the original Pocket Guide written by Joan Zimmerman under grant number HS-01540 from the National Center for Health Services Research, Department of Health, Education and Welfare. Special thanks are due Dan Schullman for his extensive commentary on this revision, and to Joan Zimmerman for her continued assistance and for permission to include her "Aerobics" program as an Appendix. Substantial contributions to both editions have been received from many members of the MUMPS community, particularly Jack Bowie, Robert Greenfield, Dan Schullman, David Sherertz, Robert Stimac, Tony Wasserman, and Jerry Wilcox. Comments and/or criticisms should be addressed to Joel Achtenberg, Washington University School of Medicine, Diabetes Research & Education Center, 100 North Euclid, St. Louis, Missouri 63108.

DATA TYPES AND VALUES

The first thing to consider is the types of data which may be manipulated in Standard MUMPS. The situation is quite simple because there is only one data type: the variable-length character string, which may consist of any of the 128 ASCII codes. [To insure portability, the current limit on MUMPS string length is 255 characters.]

A number is regarded merely as a special case of a string, and MUMPS contains a well defined rule for interpreting any string as a number. A numeric value (that is, the value of a number) may contain a decimal fraction. However, some arithmetic operations produce an integer value, which is a special case of a numeric value. Accordingly, MUMPS also contains a rule for interpreting any number (and, by inference, any string) as an integer. Relational and logical operations (see below) produce a special numeric value called a truth value. There are two truth values: the integer 0, which denotes "false"; and the integer 1, which denotes "true". The interpretation rules are discussed under OPERATORS, below.

[To insure portability the value of a number should have no more than nine significant digits. The absolute value should lie between 10^{25} and 10^{-25}, or be zero.]

VARIABLES

A variable is an entity whose value may be changed. There are three variable types: local variables, each of which is unique to a user and whose value may be inspected and/or changed only by that user; global variables, each of which may have its value inspected and/or changed by any authorized user; and special variables, whose values are changed by the MUMPS system and cannot be changed directly by a user.

Each variable is referenced by a variable name. Local and global variable names begin with either "%" or an alphabetic. Subsequent characters may be any of the alphabetics or any of the ten digits. Names may be any length. [To insure portability, however, names should be distinguishable by their first eight characters and should not contain lower-case alphabetics.] A global variable name is designated by a leading caret (^) symbol, as in^ MUG. A special variable name is denoted by a leading dollar ($) symbol, as in $TEST. Special variables are discussed in a section below.

Local and global variables may be subscripted in order to facilitate the grouping of values into sets (called arrays). For example, it might be more convenient to replace the variable names GAME, SET, and MATCH by A(1), and A(2), and A(3) respectively. In this case, only one level of subscripting is used, but more can be used if desired. For example, A(1,4,13) has three levels of subscripting. Both A(1) and A(1,4,13) are said to be descendants of A, and A(1,4,13) is also a descendant of A(1).

Variables can have any number of subscripts. Subscripts may be any of the allowed non-negative integers (that is, from 0 to 999999999).

A naked global reference is a shorthand syntax for specifying a global variable by omitting the variable name and possibly some of the subscripts. The first subscript in the subscript list of a naked global reference implicitly refers to the last subscript level of the most recent global reference. Thus, if a reference has been made to ^X(1), a subsequent naked reference to ^(2,3) would access the value of ^X(2,3). Note that "last global reference" includes <u>any</u> reference to <u>any</u> global. Use of the $NEXT function is particularly troublesome as it may leave the naked pointer with an unexpected value. (See $NEXT, below, for additional discussion).

<u>LITERALS</u>

A literal has a constant value and is interpreted directly. There are both numeric and string literals. A numeric literal has a mantissa optionally followed by the letter "E" and an exponent (e.g. 0.34, 10E5, 1.2E-7, -1456). A string literal consists of a set of characters enclosed in quotation marks. The empty string is represented by two adjacent quotes (""). If quotes are to be embedded within a string, each must be represented by two adjacent quotes ("THE ""KNOWN"" VALUE"). Such embedded quotes are counted as one character, not two, when determining string length. A string literal consists of zero or more of the 95 ASCII graphics enclosed in quotes, whereas a string value may contain any of the 128 ASCII characters.

<u>OPERATORS</u>

The operators in Standard MUMPS are grouped into seven types, as follows:

- Arithmetic Unary Operators
- Arithmetic Binary Operators
- Arithmetic Relational Operators
- String Binary Operators
- String Relational Operators
- Logical Operators
- Indirection

These groups differ in the interpretations made of their arguments. Each of the arithmetic operators takes the numeric interpretation of its arguments before performing the indicated operation. This involves taking the leftmost portion of the string which is either exponential (e.g. 86E5), decimal (-182.45) or integral (1964) in form. If no such form begins the string, the numeric interpretation is zero. The integer interpretation of any value is formed from the numeric interpretation by dropping any fraction. The logical operators interpret their arguments by first forming the numeric interpretation, if the result is 0 the interpretation is "false", otherwise, it is "true". The string operators require no special interpretation of their arguments since all data in Standard MUMPS are strings.

STRING	INTERPRETATION NUMERIC	INTEGER	TRUTH VALUE
"810"	810	810	1
"98 POUNDS"	98	98	1
"" (the empty string)	0	0	0
" 35"	0	0	0
"86+9"	86	86	1
"PAGE 10"	0	0	0
"-8.4"	-8.4	-8	1
"86E-1"	8.6	8	1
"---9"	-9	-9	1
"-0"	0	0	0

Each of the Arithmetic, String and Logical operators is described, along with examples, in the table on the following pages. Indirection, although formally an operator, is somewhat unique in its use, and is therefore discussed below in a separate section.

INDIRECTION

Indirection allows data values to be interpreted as MUMPS code. Indirection is denoted by the character @ followed by an expression. The value of the expression is substituted for the occurrence of indirection before the rest of the line is interpreted. The substitution is temporary, taking place each time the instance of indirection is encountered. This allows the same segment of MUMPS code to be executed repeatedly with differing values of the expression yielding different results. Furthermore, indirection can be nested, giving even greater flexibility. There are three basic types of indirection:

In Argument Indirection the indirection occurs in place of a command argument, and the value must be one or more complete command arguments.

```
ex.  S X(1)="STATE(I)",X(2)="CAPITAL(I)",X(3)="POPUL(I)"
     F I=1:1:50 F J=1:1:3 W $P(X(J),"(",1),": " R @X(J),!
```

reads the name, capital and population of each of the fifty states and stores them in arrays STATE, CAPITAL and POPUL.

```
ex.  S PRINTER=3,TERMINAL=0
     R !,"DEVICE TYPE? ",DEV OPEN @DEV
```

In this example, the user $enters "PRINTER" or "TERMINAL" and the proper device will be opened.

```
ex.  S USERCODE="B"
     S @("AA"_USERCODE_"=99")
```

In Name Indirection the indirection occurs in any context where a named variable can occur and the value of the indirection must be a complete variable name, possibly including subscripts.

Table of Operators

Operator Symbol	Meaning	Examples	Results & Comment
ARITHMETIC UNARY OPERATORS			
+	Takes the numeric interpretation	+2	2
		+"34A"	34
-	Takes the numeric interpretation and negates it	-A	-7 if the value of A is 7
		-"34A"	-34
ARITHMETIC BINARY OPERATORS			
+	Produces the sum	2+7	9
		A+3	The value of A plus 3
-	Produces the difference	2-7	-5
		A-B	The difference between A and B
*	Produces the product	2*7	14
/	Produces the full division	7/4	1.75
\	Division with the result truncated to an integer	7\4	1
		A\5	0 where A is greater than -5 but less than 5
#	Produces the value of the left argument modulo the right argument (i.e. it gives the remainder)	11#3	2
		11#-3	-1
		-11#3	1
		-11#-3	-2
ARITHMETIC RELATIONAL OPERATORS			
<	Less than	MINIMUM < 8	1 if the numeric value of MINIMUM is less than 8; 0 otherwise
>	Greater than	MAXIMUM > LIMIT	1 if the numeric value of MAXIMUM is greater than the numeric value of LIMIT; 0 otherwise

Table of Operators (continued)

Operator Symbol	Meaning	Examples	Results & Comment
STRING BINARY OPERATOR			
_	Concatenates	"GO"_END	"GOING" if END="ING" "GOES" if END="ES" "GO" if END=""
STRING RELATIONAL OPERATORS			
=	Equals	NAME="JONES"	1 if the value of NAME is "JONES"; 0 otherwise
[	Contains	NAME["SON"	1 if the value of NAME contains "SON"; 0 otherwise
]	Follows	NAME]"M"	1 if the value of NAME follows "M" in the ASCII collating scheme; 0 otherwise
?	Pattern matches	NAME?2A	1 if the value of NAME contains exactly two alphabetics; 0 otherwise
	For the pattern-match operator the allowed pattern codes are: A (for the 26 upper-case and 26 lower-case alphabetics); C (for the 33 control characters); E (for the entire set of 128 characters); L (for the 26 lower-case alphabetic characters); N (for the 10 numeric characters); P (for the 33 punctuation characters); U (for the 26 upper-case characters); or any string literal.	NAME?1A.A	1 if the value of NAME has one or more alphabetics; 0 otherwise

Table of Operators (concluded)

Operator Symbol	Meaning	Examples	Results & Comment
	The number of occurrences of each pattern type may be specified exactly by a preceding integer, or may be left inexact by a preceding ".". Thus, "1N" checks for exactly one numeric character and ".P" checks for any number of punctuation marks, including none. Allowed patterns may be combined into groups. For example, while ".A" checks for alphabetics and ".P" checks for punctuation, ".AP" checks for both alphabetics and punctuation; note that ".PA" has the same effect. However, literals may not be combined with pattern codes (i.e. .A"7" is not allowed).		
LOGICAL OPERATORS			
&	And	A&B	1 if A and B are true 0 if A and B are not both true
!	Or	C!D	1 if C or D is true 0 if neither C nor D is true
'	Not	'E	1 if E is false 0 if E is true
	Note that not (') may be used in conjunction with any arithmetic relational or string relational operator (e.g. A'=3, which is the same as '(A=3)).		

```
ex.  S AGE=40,RACE="WHITE",SEX="FEMALE"
     R !,"WHAT VARIABLE WOULD YOU LIKE TO SEE? ",X
     W !,X," HAS THE VALUE ",@X
```

yields RACE HAS THE VALUE WHITE if the user answers the question with "RACE".

In Pattern Indirection the indirection occurs in place of a pattern, and the value must be a pattern.

```
ex.  S X(1)="1""$""1N.N",Y(1)="MONEY"
     S X(2)="5U.U",Y(2)="WORD"
     FOR I=1,2 IF STRING?@X(I) WRITE !,Y(I) Q
```

yields "DOLLARS" if STRING is one or more digits preceded by a dollar sign, or "WORD" if STRING is five or more upper-case letters.

Note that the XECUTE command (see below) also provides a means of performing indirection.

EXPRESSIONS

The simplest expression in MUMPS is a variable, a string literal, a numeric constant, or a function (functions are discussed in a section below). Examples of each of these four types of expression are respectively:

```
VARIABLE
"LITERAL"
45.73
$LENGTH(XYZ)
```

Such simple expressions are called atomic expressions. More complicated expressions can be built up by linking a number of atomic expressions by means of the arithmetic and other types of operator. For example:

```
SUM/TOT
SEX="MALE"
"BOY"_"HOOD"
```

All MUMPS expressions are evaluated from left to right. There is a hierarchy of operators: unary operators are executed before indirection, which is executed before binary and relational operators. There is, however, no hierarchy among binary and relational operators. Parentheses can be used to modify the order of evaluation.

COMMANDS

A command defines an action to be taken. A command is usually (but not always) followed by an argument (or a series of arguments separated by commas) upon which the command acts. Most command words may be abbreviated to their initial letter, or may be fully spelled out. Note that partial abbreviations are not allowed. Thus, "B" is a legal abbreviation for "BREAK" but "BR" is not. All unspecified initial letters are reserved for future use.

Table of Commands

Command	Examples	Explanation
BREAK	B	BREAK provides an access point within MUMPS for direct mode (nonstandard) programming and debugging aids. It suspends execution until receipt of a signal (not specified) from a device. BREAK does not normally take an argument, and any arguments used are implementation specific.
CLOSE	C 4 C MT1:(devparms)	CLOSE releases the listed devices from ownership; for example, it releases device 4 or MT1. A list of implementation-specific device parameters may be placed after a device name if desired. If the current device is closed, the special variable $IO will be empty or reset to a default value, depending upon the implementation.
DO	D NEW,OLD D ^A D LINE^ROUTINE D @VARIABLE	Each of the line labels (for example, NEW and OLD) listed in the argument are found in turn and the code following each label is executed until the end (specified below) is reached. If "^" precedes a label (as in the case of ^A), the routine whose name (A) follows the "^" character is executed. Execution starts at the first line of the named routine unless a line label (for example, LINE) is specified in front of the "^". In the latter case, execution starts in the routine named ROUTINE at the line labeled LINE. Execution returns immediately to the next argument or the next command after executing a QUIT command or reaching the end of a routine (provided this was not within the scope of a subsequent DO command or a FOR command). Indirection (e.g. "@VARIABLE") can be used to begin execution of the line, or routine whose label is contained in a variable.
ELSE	E	ELSE permits conditional execution dependent upon the truth value of the special variable $TEST (discussed below). Execution of the remainder of the line to the right of the ELSE command continues only if the value of $TEST is 0 (false). If $TEST is 1 (true) any commands to the right of ELSE are not executed. ELSE does not take an argument.
FOR	F I=7,"BD" F J=2:4:20 F K=1:1 F L=10:-1:0,13,15:1	FOR specifies the repeated execution of all the commands following it on the line. There are three formats. First is the expression format, where a local variable is assigned a value such as the value 7, or "BD". Second is the range format, where a local variable is set to an initial value and then incremented in specified steps until it reaches or exceeds a limiting value; in the second example, the initial value is 2, the incremental step is 4, and the final value is 20. Note that in this case 18 is the last value of J for which the associated commands will be executed. Third is the "endless" loop format wherein only an initial value and an incremental value are specified, as shown in the third example; the "endless" loop is broken only by execution of a QUIT or a GOTO command within the instructions following the FOR command. Any of the above formats may be mixed; in the final example, we show the range format (using a negative increment), then the expression format, and finally the "endless" loop. Note that a QUIT or GOTO command terminates all forms of the FOR loop, not just the "endless" loop, and that a QUIT will terminate only the most recent in a series of nested FOR loops, whereas GOTO will terminate all active FOR loops.

Table of Commands (continued)

Command	Examples	Explanation
GOTO	G NEXT G ^ROUTINE G LINE^ROUTINE G @VARIABLE	GOTO transfers execution of code (without return) either to a line of code in the same routine (for example, G NEXT), to the start of a specified routine (for example, G ^ROUTINE), or to a particular line of code in a specified routine (for example, G LINE^ROUTINE causes execution to be transferred to the routine named ROUTINE at the line named LINE). (Note that if return of control is required, the DO command should be used.)
HALT	H	First LOCK (see below) with no arguments and CLOSE of all devices opened by this job are executed (although these are not stated explicitly). Then execution of the current process is terminated. HALT does not take an argument.
HANG	H 3 H TIME	HANG suspends execution for the number of seconds specified by each argument (such as 3 or TIME). If an argument is less than zero, execution is suspended for zero seconds. Note that HANG without an argument becomes HALT.
IF	I I A=3 I B="B",A>4 I C=4!(C=5) I SD(X),X=4	IF permits conditional execution. In its argumentless form, execution is dependent upon the value of $TEST. If the value of $TEST is 0 (false), the remainder of the line to the right of the IF is not executed; if the value of $TEST is 1 (true), execution continues normally at the next command on the line. If one argument is present (such as A=3), the argument is evaluated to be true or false and $TEST is set to 1 or 0 respectively; then the IF command is executed as in the argumentless form. A series of arguments following an IF command is treated like a series of IF commands with single arguments, the ultimate result being the same as if the arguments were "and"-ed together, except that as soon as one false argument is found, interpretation and execution of the line ceases.
KILL	K K A,NEW,^P(3) K (SAVE,A)	In its argumentless form, KILL permanently removes all existing local variables, including subscripted local variables. When an argument is specified, the local and global variables named in the argument are deleted together with all descendant variables. In the second example, local variables A and NEW are removed, as are all nodes in global ^P which have a first-level subscript of 3. The "exclusive kill" is a special form of the KILL command where the argument of the KILL command is a series of unsubscripted local variables (such as SAVE and A) contained in parentheses; all local variables are killed <u>except</u> those named and their descendants.

Table of Commands (continued)

Command	Examples	Explanation
LOCK	L L ^MUG L (A,B,^G(4)) L X123:0	LOCK is used primarily as a software convention to avoid conflicting updates of named resources (primarily global variables). In executing each argument, LOCK releases all previously specified exclusive claims to resources. If any arguments are specified, new temporary exclusive claims are established to the named resources. If another user has exclusive claims on a resource named, the current user (that is, the one executing the LOCK command) remains suspended awaiting the availability of that resource. A "timeout" can be affixed to any argument of LOCK to abort an unsuccessful wait, as mentioned above. (Note that a series of named resources must be placed in parentheses, as for L(A,B,^G(4)) but that parentheses are not required around the name for a single resource.) LOCK has no effect on the value or definition of existing variables or on the "naked indicator."
OPEN	O 4 O 7:(devparms)	The OPEN command is used to obtain exclusive ownership of a device. It does not affect the "current device" with which the routine is interacting. Implementation-specific device parameters may be placed after a device name. Ownership is relinquished upon execution of the CLOSE command. A "timeout" may be affixed to any argument to abort an unsuccessful attempt to OPEN that device.
QUIT	Q	QUIT defines an exit point following a FOR, a DO, or an XECUTE command. It does not take an argument.
READ	R VALUE,X R *A R "NAME: ",N R "WAIT? ",X:30	READ calls for data input from the current device, and the assignment of the response to a named local variable. When the argument contains an asterisk preceding a variable name, a code representing a single character is obtained. Any text and format control characters (see below) in the argument of the READ command are output on the current device. The amount of time for completion of input may be specified by affixing a timeout to an argument of READ.
SET	S NEW="BIG" S X="B",^GO(1)=9 S (I,J,K)=1	Set is the general means for explicitly assigning values to variables. When a list of variable names in parentheses is placed between the SET command and the assignment symbol ("="), each named variable is given the value following the assignment symbol.
USE	U 4 U MT+2:(devparms)	USE designates a specific device (such as device number 4 or device MT+2) as the current device for input and output, or device status. Device designators are implementation specific. Before a device can be named in the argument of a USE command, its ownership must have been established through execution of an OPEN command. As for CLOSE and OPEN, implementation-specific device parameters may be placed after a device name.

Table of Commands (concluded)

Command	Examples	Explanation
VIEW	Implementation specific	VIEW provides an access point within Standard MUMPS for the examination or modification of implementation-dependent information.
WRITE	W "HELLO" W A,B,Z W *13 W !?5,NAME W "AVG= ",SUM/TOT	WRITE specifies the output of data and format control to the current device. When an argument includes an asterisk followed by an integer value, one character whose code is the number represented by the integer is sent to the current device. The effect of this character at the device is device-dependent and implementation specific.
XECUTE	X "S A=3"	XECUTE provides a means of interpreting MUMPS code which arises during program execution. Each argument of the XECUTE command is interpreted as if it were a line of MUMPS code (without label or line start indicator).
Z	Implementation specific	Names of commands beginning with the letter Z are reserved for implementation specific extensions to Standard MUMPS.

A command and its arguments in a line of MUMPS code are separated from the next command (if any) on that line by a single space. Also, a command word is separated from its arguments by a single space. Note that this means an argumentless command is separated from the next command by exactly two spaces. In general, for any command Z with arguments A and B, Z A,B is equivalent to Z A Z B. Comments can be appended to a line by placing a semicolon in any command position (i.e., wherever a command could go). The remainder of the line is treated as a comment and is not executed.

TIMEOUTS ON COMMANDS

A timeout may be used with arguments of the LOCK, OPEN, and READ commands to specify the maximum time during which MUMPS will await the completion of the associated operation. If a timeout is to be used, it is specified by following an argument of any of these commands with a colon and then an expression whose value is an integer. For example:

```
READ ANSWER:TIME
```

will wait for up to TIME seconds to obtain ANSWER. If the numeric value following the colon is positive, that value specifies the maximum number of seconds waited for the completion of the operation. If the numeric value following the colon is zero or negative, execution continues without delay. Note that if a LOCK, OPEN, or READ command with a timeout specification is satisfied before the time runs out, the special variable $TEST (discussed below) is set to 1; otherwise $TEST is set to 0. If no timeout is specified, $T is not affected and execution of the command proceeds after the condition associated with the command (for example, termination of the input message for the READ command) is satisfied. Therefore, execution can be suspended indefinitely.

POST-CONDITIONALS ON COMMANDS AND ARGUMENTS

The IF command may be used to place a condition upon whether or not the remainder of the line following the IF command is executed. Alternatively, an individual condition may be placed upon most commands by appending a colon and then a truth-valued expression to a command word. This may be done for all commands except ELSE, FOR, and IF. For example,

```
SET:A=3 NEXT=47 GOTO LINE
```

means that NEXT has the value of 47 assigned only if A has the value 3, but the GOTO command is always executed. The first command is said to be post-conditionalized. It is executed only if the expression immediately following the command and colon is true. Note that unlike the IF command, only the post-conditionalized command is affected; all commands to its right are still executed as they would be normally. Also, the value of $TEST is not affected.

The commands DO, GOTO, and XECUTE permit post-conditionalization of arguments and/or post-conditionalization of the command itself. This means that an argument of any of these commands may be followed by a colon and then a truth-valued expression. If the expression is true, that argument is used. Otherwise, it is not. For example:

```
DO FIND:A=100,NEXT
```

will execute the code beginning at the line labeled FIND only if A equals 100, but will always execute the code beginning at the line labeled NEXT. Thus, only an argument which has been post-conditionalized is affected. All arguments and further command-argument pairs to the right of a post-conditionalized argument are executed as they would be normally.

The following example illustrates post-conditionalization of both command and argument:

```
DO:B="OK" FIND:A=3,NEXT
```

FUNCTIONS

Each function is designated by an initial character of "$", and has a unique name which may be abbreviated to its initial letter. This specification of the function is followed by one or more expressions in parentheses. The first expression generally signifies the string or number which is to be examined or manipulated. Any subsequent expressions qualify the function's effect. All unspecified initial letters are reserved for future use.

Some functions restrict the type of expressions that can be used. The following codes will be used to indicate such restrictions. In each case "n" represented the position in which the expression is used.

En	general (unrestricted) expressions
In	Integer-valued expressions
Tn	Truth-valued expressions
Nn	Numeric valued expressions
L	Labels
VN	Variable names

SPECIAL VARIABLES

Each special variable is denoted by the initial character of "$" and has a unique name which may be abbreviated to its initial letter. The value of a special variable may be used as part of any general expression. Note, however, that a user is not permitted to assign a value to any special variable. All unspecified initial letters are reserved for future use.

Special Variable	Explanation
$HOROLOG	$HOROLOG provides the date and time in a single two-part string. The two parts are separated by a comma. The first part is the number of days since 31 December 1840, and the second part is the number of seconds since midnight. "0,0" is the first second of 31 December 1840.
$IO	$IO provides the unique identification of the current input/output device.

Table of Functions

Function Name and Syntax	Explanation	Examples	Results
$ASCII(E1) $ASCII(E1,I2)	$ASCII selects a character of a string and returns its ASCII code. E1 specifies the string of interest. If I2 is not specified explicitly, it is assumed to have a value of 1. The character in position I2 of string E1 is extracted and translated into the integer which represents it in the ASCII sequence.	$A("ABC") $A("ABC",1) $A("ABC",2) $A(X,2) $A(" ")	65 65 66 66 where X="ABC" -1
$CHAR(I1) $CHAR(I1,I2) $CHAR(I1,I2, . . .)	$CHAR translates a set of integers into a string of characters whose ASCII codes are those integers. I1,I2 and so on are the expressions whose values are interpreted as the integers (which must be in the range 0-127, inclusive). A negative integer yields an empty string; values greater than 127 yield an error.	$C(65) $C(66) $C(65,66)	"A" "B" "AB"
$DATA(VN)	$DATA returns an integer specifying whether a variable is defined or undefined. VN is the name of the local or global variable of interest. The values returned are 0 if VN is not defined, 1 if VN has a value but has no associated descendant array members, 10 if VN does not have a value but has at least one descendant and 11 if VN has both a value and at least one descendant. Note: VN might not have a descendant even when $D(VN) > 9 under certain circumstances; e.g. all descendants have been killed.	$D(Y) $D(Y) $D(^A(1,5))	0 if Y is undefined 1 if Y has a value but has no descendant 0, 1, 10 or 11
$EXTRACT(E1,I2) $EXTRACT(E1,I2,I3)	$EXTRACT returns a character or substring of string E1 beginning at character position I2 and ending at character position I3 (at I2 if I3 is not specified) or at the end of the string, whichever occurs first.	$E("ABC",2) $E(VAR,3,5) $E(VAR,3,999) $E(S,90) $E("XYZ",-6,2)	"B" "CDE" if VAR="XXCDE". "10A" if VAR="B910A". " " if S is less than 90 characters long. "XY"

Table of Functions (continued)

Function Name and Syntax	Explanation	Examples	Results
$FIND(E1,E2) $FIND(E1,E2,I3)	$FIND returns an integer which specifies the end position plus one of a substring (E2) within a string (E1). If a third expression, I3, is specified, the search in E1 for E2 begins at the character in position I3. Otherwise the search begins at position 1. If the portion of E1 beginning at I3 (or 1) is the empty string or does not contain E2 the result is 0 (zero). If E2 is null (but not the portion of E1 beginning at I3 or 1) the result is I3.	$F("ABC","B") $F("ABCABC","A",3)	3 5
$JUSTIFY(E1,I2) $JUSTIFY(N1,I2,I3)	$JUSTIFY with 2 arguments returns the value of expression E1, right-justified within a field of size E2. However, if I2 is less than or equal to the length of E1 the value of E1 is returned unchanged. When 3 arguments are given the numeric interpretation of the first argument (N1) is formed, this value is then rounded to have I3 decimal places after the decimal point (including possible trailing zeros) and the result is right-justified in a field of I2 spaces.	$J(39,3) $J(39,4,1) $J(-5/3,0,0)	" 39" "39.0" "-2"
$LENGTH(E1)	$LENGTH returns the number of characters in string E1, that is, it returns the string length.	$L("ABC") $L(X) $L(X)	3 4 where X="ABCD". 0 where X=" ".
$NEXT(VN)	$NEXT returns the next numerically lowest existing subscript above that specified in VN. If there is no higher subscript, the result returned is −1. Note that VN need not exist.	$N(A(1)) $N(A(-1)) $N(^MUG(4))	2 if A(2) is defined (either pointer or datum). −1 if no subscripts exist. 6 if ^MUG(6) exists and ^MUG(5) does not.
$PIECE(E1,E2,I3) $PIECE(E1,E2,I3,I4)	$PIECE inspects a specified string (E1) and takes from it that substring which lies between two specified occurrences of a particular dividing string (E2). The resulting substring begins immediately after the I3 minus one (cont'd.)	$P("A.BX.Y",".",2) $P("76A666","6",1,4) $P(ST,".",2) $P(ST,".",1)	"BX" "76A66" " " if ST does not contain a period. ST if ST does not contain a period.

Table of Functions (concluded)

Function Name and Syntax	Explanation	Examples	Results
	occurrence of the dividing string and ends immediately before the I4th occurrence (or I3rd if I4 is unspecified).		
$RANDOM(I1)	$RANDOM returns an integer in the range 0 through I1 minus one. Each integer value has equal probability of occurrence.	SR(2) SR(100)	0 or 1. An integer between 0 and 99.
$SELECT(T1:E1,T2:E2, . . ., Tn:En)	$SELECT evaluates the left-hand expression in each of a series of expression pairs, one at a time in left-to-right order, until one is found with a truth value of 1 (true), then returns the value of the right-hand expression of that pair. In each pair, the truth-valued expression (Tn) on the left is separated from its "result" expression on the right (En) by a colon. There may be one or more pairs in the series, but at least one of the truth-valued expressions must be true.	$S(A=3:5,1:0) $S(X=7:"HI",1:"BYE") $S(Z=1:1,Z < 0:0,1:2)	5 if A has the value of 3. "BYE" if X has a value not equal to 7. 0 if X is negative.
$TEXT(L1) $TEXT(L1+I2) $TEXT(+I1)	$TEXT returns the text of a specified line of the current routine, including the line-start indicator and label. Lines may be referenced by label (L1), by a positive offset from a label (L1+I2), or by a positive offset from the beginning of the routine (+I1). If the specified line does not exist, the empty string is returned.	$T(LINE) $T(A+3) $T(+2)	Returns the text of the line labeled LINE. Returns the text of the third line after the line labeled A. Returns the second line of the routine.
$VIEW (argument syntax specified by the implementor)	$VIEW is an implementation-specific function available (at the option of the implementor) for providing implementation-specific data.	–	–
$Z_	Any additional function not included in the Standard should begin with "$Z".	–	–

$JOB — Each executing MUMPS process has its own unique job number, a positive integer, which is the value of $JOB.

$STORAGE — $STORAGE is the number of unused characters remaining in the user's partition.

$TEST — $TEST contains the last computed truth value. The value is set by the execution of the most recent IF command containing an argument, or by an OPEN, LOCK, or READ with a timeout.

$X — $X is the column or horizontal position at which the carriage (for a printing device) or the cursor (for a video terminal) lies for the current device. The first column is defined as column 0, the second column as column 1, and so on. Therefore, $X is 0 at the start of a line. After the first character in a line has been written, $X is 1. $X is initialized to zero by carriage return, and incremented by 1 for graphics.

$Y — $Y is the vertical position on the current device. The first line is defined as line 0, the second as line 1, and so on. $Y is initialized to zero by a form feed, and incremented by 1 for each line feed.

$Z_ — Each nonstandard special variable should begin with "$Z".

FORMAT CONTROL

The following characters are used for format control during data display:

? specifies tab (right shift) to the column specified by the integer value of the expression following the question mark. For example, ?X tabs to column X. Remember (see $X above) that MUMPS regards the first column as column 0, the second as column 1, and so on. If X $X no shift takes place.

! specifies carriage-return and line-feed, or other similar new-line operation.

specifies a form-feed or new page, or similar operation.

LINES AND ROUTINES

A routine in MUMPS consists of an ordered series of lines. Lines are NOT free-format. Each consists of a line-start indicator (specified by the implementor), preceded by an optional line label, followed by zero or more commands and associated arguments, followed by an optional comment. Multiple commands in a line are separated from each other by exactly one

space, except that argumentless commands are separated from subsequent commands by exactly two spaces (where one blank indicates a null argument). Comments may be inserted into a line by placing a semicolon after the last argument or argumentless command. The remainder of the line is treated as a comment and is not executed.

A line label may begin with an alphabetic or with the "%" character. The subsequent characters may be any of the 26 alphabetics or any of the ten numeric digits. (To promote program portability alphabetic characters in labels should be upper-case only). Alternatively, the label may be entirely numeric digits characters (i.e., an integer). The label may be any length, but only the first eight characters are distinguished. Labels should be unique within a routine.

[For reasons of program portability, the length of a line of stored MUMPS code is currently limited to 255 characters, including the label, line start indicator, code and comments. There is no explicit limit on the number of lines in a MUMPS routine, but the routine including all local variables and temporary result storage should not exceed 4,000 characters.]

Note that the grouping of multiple commands into a single line is more than simple convenience. Commands such as FOR, IF, and ELSE treat the line as a meaningful unit.

APPENDIX I: Table of ASCII Characters

The character notation is that used in ANS X3.4-1968. The code values are those which appear as values of the $ASCII function and as arguments of the $CHAR function.

Code	Character	Code	Character	Code	Character	Code	Character
0	NUL	32	SP	64	@	96	
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	DEL

APPENDIX II: Sample Routine

The program which follows was written by Dr. Joan Zimmerman to illustrate many of the features of a Standard MUMPS routine. Note that the Line Start Indicator is represented in this printout by tabulation to column 6.

```
ZEX2  ;DDS & ZIM;28 MAR 79;COOPER'S AEROBICS POINT SYSTEM (1970)

START KILL  SET SUM=0
      READ "DID YOU RUN OR WALK TODAY? ",ANS IF ANS="Y" DO RUN
      READ "DID YOU SWIM TODAY? ",ANS IF ANS="Y" DO SWIM
      READ "DID YOU BICYCLE TODAY? ",ANS IF ANS="Y" DO CYCLE
      HALT

RUN   SET MAX=100,AVERAGE=2 DO DIST IF MILES=0 QUIT
      SET MIN=3.7,AVERAGE=8 DO TIME
      IF (MILES<2)&(T'<20) SET POINTS=0
      ELSE  SET POINTS=$SELECT(T<6.5:6,T<8:5,T<10:4,T<12:3,T<14.5:2,T<20:1,1:.5)*MILE
      DO ACCUM
      GOTO RUN

SWIM  ;CODE TO BE WRITTEN
      QUIT

CYCLE SET MAX=1000,AVERAGE=9 DO DIST IF MILES=0 QUIT
      SET MIN=.5,AVERAGE=4 DO TIME
      IF (MILES<5)&(T'<6) SET POINTS=0
      ELSE  SET POINTS=$SELECT(T<3:1.5,T<4:1,T<6:.5,1:.22)*MILES
      DO ACCUM
      GOTO CYCLE

DIST  READ "NUMBER OF MILES COMPLETED: ",MILES
      IF (MILES<0)!(MILES>MAX) WRITE "MOST UNLIKELY ... TRY AGAIN",! GOTO DIST
      IF MILES>AVERAGE WRITE "    VERY GOOD",!
      QUIT

TIME  READ "HOW MANY MINUTES DID THAT TAKE? ",TIM
      IF TIM'>0 WRITE "YOU CAN'T DO THINGS THAT QUICKLY!",! GOTO TIME
      SET T=TIM/MILES IF T<AVERAGE WRITE "    THAT'S FAST!",!
      IF T<MIN WRITE "ACTUALLY, THAT'S UNBELIEVABLE ... TRY AGAIN",! GOTO TIME
      QUIT

ACCUM SET SUM=SUM+POINTS
      WRITE "YOU JUST ADDED ",POINTS," POINT" IF POINTS'=1 WRITE "S"
      WRITE " AND YOUR TOTAL IS ",SUM,!,"ENTER MORE",!!
      QUIT
```

APPENDIX III: IMPLEMENTATION - SPECIFIC FEATURES.

SPACE IS PROVIDED BELOW FOR VENDORS OR USERS TO DEFINE IMPLEMENTATION-SPECIFIC FEATURES OF THEIR SYSTEM.

Line Start Indicator

End of Line

Read (Response) Terminator

Commands (begin with Z)

Functions (begin with $Z)

Special Variables (begin with $Z)

Arguments of BREAK command

VIEW Command

Device Specifiers

Device Parameters

MUMPS USER'S GROUP
4321 Hartwick Rd. #308
College Park, MD 20740